Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>$P$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j < m$ we have that $P[j] = T[i + j]

(our strings are zero-indexed)
Exact pattern matching

**Input** A text string \(T\) (length \(n\)) and a pattern string \(P\) (length \(m\))

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
T & a & b & c & b & a & b & a & b & a & c & a & b & a \\
\end{array}
\]

\[
P & a & b & a \checkmark \\
\]

\[m\]

**Goal:** Find all the locations where \(P\) matches in \(T\)

\(P\) matches at location \(i\) iff for all \(0 \leq j < m\) we have that \(P[j] = T[i + j]\)

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>$P$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff
for all $0 \leq j < m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>$P$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j < m$ we have that $P[j] = T[i + j]$

(our strings are zero-indexed)
Exact pattern matching

Input: A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

Goal: Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j < m$ we have that $P[j] = T[i + j]$ (our strings are zero-indexed)
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

Goal: Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j < m$ we have that $P[j] = T[i + j]$

(our strings are zero-indexed)
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

```
T  a  b  c  b  a  b  a  b  a  c  a  b  a
P  a  b  a
```

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j < m$ we have that $P[j] = T[i+j]$  
(our strings are zero-indexed)

- A naive algorithm takes $O(nm)$ time
Exact pattern matching

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

$T$

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$P$

---

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j < m$ we have that $P[j] = T[i + j]$

*(our strings are zero-indexed)*

- A naive algorithm takes $O(nm)$ time
- Many $O(n)$ time algorithms are known (for example KMP)
Exact pattern matching: another solution

\[ T \begin{array}{cccccccc}
  a & b & c & b & a & b & a & b & a \\
\end{array} \]

\[ P \begin{array}{c}
  6 & a & b & a \\
\end{array} \]

---

\[ m \]
Exact pattern matching: another solution

In the remainder, we assume that the symbols are numbers... 
(if they aren't just use their bit-representation)
Exact pattern matching: another solution

In the remainder, we assume that the symbols are numbers…

(if they aren’t just use their bit-representation)
Exact pattern matching: another solution

\[
\begin{array}{cccccccccccc}
& 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
T & 1 & 2 & 3 & 2 & 1 & 2 & 1 & 2 & 1 & 3 & 1 & 2 & 1 \\
P & 6 & 1 & 2 & 1 \\
\end{array}
\]
Consider the following expression,

\[ d(i) = \sum_{j=0}^{m-1} (T[i+j] - P[j])^2 \]
Consider the following expression,

\[
d(i) = \sum_{j=0}^{m-1} (T[i+j] - P[j])^2
\]

When do exact matches occur?
Consider the following expression,

\[ d(i) = \sum_{j=0}^{m-1} (T[i + j] - P[j])^2 \]

When do exact matches occur? When \( d(i) = 0 \)
Consider the following expression,

\[ d(i) = \sum_{j=0}^{m-1} (T[i+j] - P[j])^2 \]

When do exact matches occur? When \( d(i) = 0 \)

\[ d(i) = 0 \text{ iff } T[i \ldots i+m-1] = P. \]
By multiplying out the brackets we have that,
Exact pattern matching: computing terms

By multiplying out the brackets we have that,

\[ d(i) = \sum_{j=0}^{m-1} (T[i+j] - P[j])^2 = \]
Exact pattern matching: computing terms

By multiplying out the brackets we have that,

\[ d(i) = \sum_{j=0}^{m-1} (T[i+j] - P[j])^2 = \]

\[ \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]
Exact pattern matching: computing terms

By multiplying out the brackets we have that,

\[ d(i) = \sum_{j=0}^{m-1} (T[i+j] - P[j])^2 = \]

\[ \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

How hard is it to compute each of these terms?

(for all \(0 \leq i < n\))
Exact pattern matching: computing terms

d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

The first term depends only on the pattern
Exact pattern matching: computing terms

$$d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j]$$

The first term depends only on the pattern
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The first term depends only on the pattern.
It is the same for all \( i \) so can be computed in \( O(m) \) time.
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The second term depends only on the text
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The **second term** depends only on the text
It can be computed by a sliding window in \( O(n) \) time. . .
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The second term depends only on the text. It can be computed by a sliding window in \( O(n) \) time.

\[ \sum_{j=0}^{m-1} T[i+j]^2 = \sum_{j=0}^{m-1} (T[(i-1)+j]^2) + T[i+m-1]^2 - T[i-1]^2 \]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The second term depends only on the text. It can be computed by a sliding window in \( O(n) \) time.

\[ \sum_{j=0}^{m-1} T[i+j]^2 = \sum_{j=0}^{m-1} (T[(i-1)+j]^2) + T[i+m-1]^2 - T[i-1]^2 \]

the old value \quad add this \quad subtract this
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The second term depends only on the text.
It can be computed by a sliding window in \( O(n) \) time...

\[
\sum_{j=0}^{m-1} T[i+j]^2 = \sum_{j=0}^{m-1} (T[(i-1)+j]^2) + T[i+m-1]^2 - T[i-1]^2
\]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

The second term depends only on the text. It can be computed by a sliding window in \( O(n) \) time.

\[ \sum_{j=0}^{m-1} T[i + j]^2 = \sum_{j=0}^{m-1} (T[(i-1)+j]^2) + T[i+m-1]^2 - T[i-1]^2 \]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The second term depends only on the text. It can be computed by a sliding window in \( O(n) \) time...

\[ \sum_{j=0}^{m-1} T[i+j]^2 = \sum_{j=0}^{m-1} (T[(i-1)+j]^2) + T[i+m-1]^2 - T[i-1]^2 \]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The third term looks more tricky...
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The third term looks more tricky…
Luckily, we have a tool for this… cross-correlation
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

The third term looks more tricky…

Luckily, we have a tool for this… cross-correlation

\[ (T \otimes P)[i] = \sum_{j=0}^{m-1} P[j]T[i + j] \]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

The third term looks more tricky…

Luckily, we have a tool for this… cross-correlation

\[ (T \otimes P)[i] = \sum_{j=0}^{m-1} P[j]T[i + j] \]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The third term looks more tricky…

Luckily, we have a tool for this… cross-correlation

\[
(T \otimes P)[i] = \sum_{j=0}^{m-1} P[j]T[i+j]
\]

\[
\begin{array}{cccccc}
2 & 1 & 2 & 1 & 2 & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
\times & \times & \times \\
1 & 2 & 1 \\
\end{array}
\]

\[
(1 \times 1) + (2 \times 2) + (1 \times 1) = 6
\]
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

The third term looks more tricky . . .

Luckily, we have a tool for this... cross-correlation

\[ (T \otimes P)[i] = \sum_{j=0}^{m-1} P[j]T[i + j] \]

\[
\begin{array}{cccccc}
2 & 1 & 2 & 1 & 2 & 1 & 3 \\
\times & \times & \times \\
1 & 2 & 1
\end{array}
\]

\[(1 \times 1) + (2 \times 2) + (1 \times 1) = 6\]

We can compute \((T \otimes P)\) via the FFT.
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

The third term looks more tricky…

Luckily, we have a tool for this... cross-correlation

\[ (T \otimes P)[i] = \sum_{j=0}^{m-1} P[j]T[i+j] \]

We can compute \((T \otimes P)\) via the FFT.

\[ \begin{array}{cccccc}
2 & 1 & 2 & 1 & 2 & 1 & 3 \\
\hline
1 & 2 & 1 & 1 & 2 & 1 & 1
\end{array} \]

\((1 \times 1) + (2 \times 2) + (1 \times 1) = 6\)
Exact pattern matching: computing terms

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

The third term \textit{looks more tricky}... 

Luckily, we have a tool for this... \textit{cross-correlation}

\[(T \otimes P)[i] = \sum_{j=0}^{m-1} P[j]T[i + j] \]

\begin{array}{c}
\begin{array}{cccc}
2 & 1 & 2 & 1 \\
1 & 2 & 1 & 3 \\
\end{array} \\
\begin{array}{c}
\times+\times+\times \\
\begin{array}{c}
1 \\
2 \\
1 \\
\end{array}
\end{array}
\end{array}

(1 \times 1) + (2 \times 2) + (1 \times 1) = 6

We can compute \((T \otimes P)\) via the FFT. (Fast Fourier Transform)

We obtain \((T \otimes P)[i]\) for all \(i\) in \(O(n \log m)\) time (we won’t cover this result in this course)
Exact pattern matching: putting it all together

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j] \]

Algorithm summary

Compute the first term directly \((O(m)\) time)
Exact pattern matching: putting it all together

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

Algorithm summary

Compute the first term directly \((O(m) \text{ time})\)
Compute the second term via a sliding window \((O(n) \text{ time})\)
Exact pattern matching: putting it all together

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

Algorithm summary

Compute the first term directly \(O(m)\) time
Compute the second term via a sliding window \(O(n)\) time
Compute the third term via FFTs \(O(n \log m)\) time
Exact pattern matching: putting it all together

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

**Algorithm summary**

- Compute the first term directly \(O(m)\) time
- Compute the second term via a sliding window \(O(n)\) time
- Compute the third term via FFTs \(O(n \log m)\) time
- Find all \(i\) such that \(d(i) = 0\) using the values found \(O(n)\) time
Exact pattern matching: putting it all together

\[
    d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i+j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i+j]
\]

Algorithm summary

Compute the first term directly \((O(m) \text{ time})\)
Compute the second term via a sliding window \((O(n) \text{ time})\)
Compute the third term via FFTs \((O(n \log m) \text{ time})\)
Find all \(i\) such that \(d(i) = 0\) using the values found \((O(n) \text{ time})\)

*Overall, we obtain a time complexity of \(O(n \log m)\).*
Exact pattern matching: putting it all together

\[ d(i) = \sum_{j=0}^{m-1} P[j]^2 + \sum_{j=0}^{m-1} T[i + j]^2 - 2 \sum_{j=0}^{m-1} P[j]T[i + j] \]

Algorithm summary

- Compute the first term directly (\(O(m)\) time)
- Compute the second term via a sliding window (\(O(n)\) time)
- Compute the third term via FFTs (\(O(n \log m)\) time)
- Find all \(i\) such that \(d(i) = 0\) using the values found (\(O(n)\) time)

Overall, we obtain a time complexity of \(O(n \log m)\).

Why would you do this!? KMP takes \(O(n)\) time.
Pattern matching with wildcards

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

$T$ | $a$ | $b$ | $c$ | ? | $a$ | $b$ | $a$ | $a$ | ? | ? | $c$ | $a$ | $a$  
---|---|---|---|---|---|---|---|---|---|---|---|---|---|---
$P$ | $a$ | $b$ | ? | $a$  

the $?$ symbol matches any single character

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff
for all $0 \leq j < m$ we have that $P[j] = T[i + j]$  
or $P[j] = ?$ or $T[i + j] = ?$
Pattern matching with wildcards

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

$$T \quad a \quad b \quad c \quad ? \quad a \quad b \quad a \quad a \quad ? \quad ? \quad c \quad a \quad a$$

$$P \quad 4 \quad a \quad b \quad ? \quad a$$

*the ? symbol matches any single character*

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j < m$ we have that $P[j] = T[i + j]$

or $P[j] = ?$ or $T[i + j] = ?$
Pattern matching with wildcards

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

$T$ = [a b c ? a b a a ? ? c a a]

$P$ = [a b ? a]

The ? symbol matches any single character

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

For all $0 \leq j < m$ we have that $P[j] = T[i + j]$ or $P[j] = ?$ or $T[i + j] = ?$
Pattern matching with wildcards

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

$T$:

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>?</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>?</td>
<td>?</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

$P$:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>

Goal: Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j < m$ we have that $P[j] = T[i + j]$

or $P[j] = ?$ or $T[i + j] = ?$

*the ? symbol matches any single character*
Pattern matching with wildcards

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>$n$</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>?</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>?</td>
<td>?</td>
<td>c</td>
</tr>
<tr>
<td>$P$</td>
<td>0</td>
<td>a</td>
<td>b</td>
<td>?</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*the ? symbol matches any single character*

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff for all $0 \leq j < m$ we have that $P[j] = T[i + j]$ or $P[j] = ?$ or $T[i + j] = ?$

- A naive algorithm takes $O(nm)$ time
Pattern matching with wildcards

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
T & a & b & c & ? & a & b & a & a & ? & ? & c & a & a \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
P & a & b & ? & a & & & & & & & & & \\
\end{array}
\]

*the ? symbol matches any single character*

**Goal:** Find all the locations where $P$ matches in $T$

$P$ matches at location $i$ iff

for all $0 \leq j < m$ we have that $P[j] = T[i + j]$

or $P[j] = ?$ or $T[i + j] = ?$

- A naive algorithm takes $O(nm)$ time

*We can do better using cross-correlations…*
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers... 

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as *positive* integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers... 

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\hline
T' & 1 & 2 & 3 & 0 & 1 & 2 & 1 & 1 & 0 & 0 & 3 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 0 & 1 \\
\hline
P' & m \\
\end{array}
\]

*call these P' and T'*
Pattern matching with wildcards: a simple solution

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
n & 1 & 2 & 3 & 0 & 1 & 2 & 1 & 1 & 0 & 0 & 3 & 1 & 1 \\
\end{array}
\]

\[T' \]

\[\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c}
0 & 1 & 2 & 0 & 1 \\
\hline
m & \end{array}\]

Again we rewrite the symbols as integers... call these \(P'\) and \(T'\)

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

Consider the following expression,

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2 \]
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers... call these $P'$ and $T'$
- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as *positive* integers

Consider the following expression,

$$d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i+j](T'[i+j] - P'[j])^2$$

When do exact matches occur?
Pattern matching with wildcards: a simple solution

Consider the following expression,

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2 \]

When do exact matches occur?

\[ d_w(i) = 0 \text{ iff } P \text{ matches } T[i \ldots i + m - 1] \]
Pattern matching with wildcards: a simple solution

Consider the following expression,

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2 \]

\[ 0 \text{ iff } P[j] = ? \]

When do exact matches occur?

\[ d_w(i) = 0 \text{ iff } P \text{ matches } T[i \ldots i + m - 1] \]
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers...
- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

Consider the following expression,

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i+j](T'[i+j] - P'[j])^2 \]

When do exact matches occur?

\[ d_w(i) = 0 \text{ iff } P \text{ matches } T[i \ldots i + m - 1] \]
Consider the following expression,

\[
d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j] (T'[i + j] - P'[j])^2
\]

When do exact matches occur?

\[
d_w(i) = 0 \text{ iff } P \text{ matches } T[i \ldots i + m - 1]
\]
Pattern matching with wildcards: a simple solution

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

\[ T' = \begin{array}{cccccccccccc}
1 & 2 & 3 & 0 & 1 & 2 & 1 & 1 & 0 & 0 & 3 & 1 & 1 \\
\end{array} \]

\[ P' = \begin{array}{c}
1 & 2 & 0 & 1 \\
\end{array} \]

\[ \mbox{Call these } P' \mbox{ and } T' \]

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

Consider the following expression,

\[
d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2
\]

When do exact matches occur?

\[
d_w(i) = 0 \mbox{ iff } P \mbox{ matches } T[i \ldots i + m - 1]
\]
Pattern matching with wildcards : a simple solution

Consider the following expression,

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2 \]

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

When do exact matches occur?

\[ d_w(i) = 0 \text{ iff } P \text{ matches } T[i \ldots i + m - 1] \]
Pattern matching with wildcards: a simple solution

Again we rewrite the symbols as integers... call these $P'$ and $T'$

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

Consider the following expression,

$$d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2$$

always $\geq 0$

0 iff $P[j] = ?$ or $T[i+j] = ?$ or $(T[i+j] = P[j])$

When do exact matches occur?

$$d_w(i) = 0 \text{ iff } P \text{ matches } T[i \ldots i + m - 1]$$
Pattern matching with wildcards: a simple solution

When do exact matches occur?

\[ d_w(i) = 0 \text{ iff } P \text{ matches } T[i \ldots i + m - 1] \]

Consider the following expression,

\[
d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2
\]

0 iff \( P[j] = ? \) or \( T[i+j] = ? \) or \( (T[i+j] = P[j]) \)

Again we rewrite the symbols as integers... call these \( P' \) and \( T' \)

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

the non-negative terms can't cancel

Consider the following expression,

\[
d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2
\]

Always \( \geq 0 \)

The non-negative terms can't cancel

Again we rewrite the symbols as integers... call these \( P' \) and \( T' \)

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

Consider the following expression,

\[
d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2
\]

Always \( \geq 0 \)

The non-negative terms can't cancel

Again we rewrite the symbols as integers... call these \( P' \) and \( T' \)

- the wildcard symbol (?) must be encoded as 0
- all other symbols are encoded as positive integers

Consider the following expression,

\[
d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j](T'[i + j] - P'[j])^2
\]

Always \( \geq 0 \)

The non-negative terms can't cancel
Pattern matching with wildcards: computing terms

By multiplying out the brackets we have that,
Pattern matching with wildcards: computing terms

By multiplying out the brackets we have that,

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i+j](T'[i+j] - P'[j])^2 = \]
Pattern matching with wildcards: computing terms

By multiplying out the brackets we have that,

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j] T'[i + j] (T'[i + j] - P'[j])^2 = \]

\[ \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] + \sum_{j=0}^{m-1} P'[j] T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]
Pattern matching with wildcards: computing terms

By multiplying out the brackets we have that,

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j] T'[i+j] (T'[i+j] - P'[j])^2 = \]

\[ \sum_{j=0}^{m-1} P'[j]^3 T'[i+j] + \sum_{j=0}^{m-1} P'[j] T'[i+j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i+j]^2 \]

How hard is it to compute each of these terms?

(for all \( 0 \leq i < n \))
Pattern matching with wildcards: computing terms

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i+j] + \sum_{j=0}^{m-1} P'[j] T'[i+j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i+j]^2 \]
Pattern matching with wildcards: computing terms

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i+j] + \sum_{j=0}^{m-1} P'[j] T'[i+j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i+j]^2 \]

Consider the first term...
Consider the first term...
Pattern matching with wildcards: computing terms

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] + \sum_{j=0}^{m-1} P'[j] T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]

Consider the first term...

Construct a new pattern \( P_3 \) where \( P_3[j] = P'[j]^3 \) in \( O(m) \) time
Pattern matching with wildcards: computing terms

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i+j] + \sum_{j=0}^{m-1} P'[j] T'[i+j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i+j]^2 \]

Consider the first term...

Construct a new pattern \( P_3 \) where \( P_3[j] = P'[j]^3 \) in \( O(m) \) time

\[ \sum_{j=0}^{m-1} P'[j]^3 T'[i+j] = \sum_{j=0}^{m-1} P_3[j] T'[i+j] \]
Pattern matching with wildcards: computing terms

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i+j] + \sum_{j=0}^{m-1} P'[j]T'[i+j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i+j]^2 \]

Consider the first term...

Construct a new pattern \( P_3 \) where \( P_3[j] = P'[j]^3 \) in \( O(m) \) time

\[ \sum_{j=0}^{m-1} P'[j]^3 T'[i+j] = \sum_{j=0}^{m-1} P_3[j]T'[i+j] = (T' \otimes P_3)[i] \]
Pattern matching with wildcards: computing terms

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] + \sum_{j=0}^{m-1} P'[j] T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]

Consider the first term...

Construct a new pattern \( P_3 \) where \( P_3[j] = P'[j]^3 \) in \( O(m) \) time

\[ \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] = \sum_{j=0}^{m-1} P_3[j] T'[i + j] = (T' \otimes P_3)[i] \]

Compute \( (T' \otimes P_3) \) (for all \( i \)) in \( O(n \log m) \) time
Pattern matching with wildcards: computing terms

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] + \sum_{j=0}^{m-1} P'[j] T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]

Consider the first term...

Construct a new pattern \( P_3 \) where \( P_3[j] = P'[j]^3 \) in \( O(m) \) time

\[ \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] = \sum_{j=0}^{m-1} P_3[j] T'[i + j] = (T' \otimes P_3)[i] \]

Compute \( (T' \otimes P_3) \) (for all \( i \)) in \( O(n \log m) \) time

Both the other terms can be computed analogously
Pattern matching with wildcards: putting it all together

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] + \sum_{j=0}^{m-1} P'[j] T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]

Algorithm summary

Compute the pattern and text transformations \((O(n)\) time)
Pattern matching with wildcards: putting it all together

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] + \sum_{j=0}^{m-1} P'[j] T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]

Algorithm summary

Compute the pattern and text transformations (\(O(n)\) time)

Compute each term using cross-correlations (\(O(n \log m)\) time)
Pattern matching with wildcards: putting it all together

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] + \sum_{j=0}^{m-1} P'[j] T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]

**Algorithm summary**

1. Compute the pattern and text transformations \( (O(n) \text{ time}) \)
2. Compute each term using cross-correlations \( (O(n \log m) \text{ time}) \)
3. Find all \( i \) such that \( d_w(i) = 0 \) using the values found \( (O(n) \text{ time}) \)
Pattern matching with wildcards: putting it all together

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]^3 T'[i + j] + \sum_{j=0}^{m-1} P'[j] T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]

Algorithm summary

Compute the pattern and text transformations (\(O(n)\) time)

Compute each term using cross-correlations (\(O(n \log m)\) time)

Find all \(i\) such that \(d_w(i) = 0\) using the values found (\(O(n)\) time)

Overall, we obtain a time complexity of \(O(n \log m)\).
Pattern matching with wildcards: putting it all together

\[ d_w(i) = \sum_{j=0}^{m-1} P'[j]T'[i + j] + \sum_{j=0}^{m-1} P'[j]T'[i + j]^3 - 2 \sum_{j=0}^{m-1} P'[j]^2 T'[i + j]^2 \]

Algorithm summary

- Compute the pattern and text transformations (\(O(n)\) time)
- Compute each term using cross-correlations (\(O(n \log m)\) time)
- Find all \(i\) such that \(d_w(i) = 0\) using the values found (\(O(n)\) time)

Overall, we obtain a time complexity of \(O(n \log m)\).

- this is the best known algorithm for this problem.
Pattern matching with mismatches

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

- $T = a\ b\ c\ d\ a\ b\ a\ a\ d\ a\ c\ a\ a$
- $P = a\ b\ d\ a$

**Goal:** For all $i$, output, $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches...*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>c</td>
</tr>
</tbody>
</table>

$T$

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>d</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$P$

$\text{Ham}(4) = 1$

**Goal:** For all $i$, output, $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches...*  
  i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
T & a & b & c & d & a & b & a & a & d & a & c & a & a \\
\hline
P & a & b & d & a \\
\hline
\end{array}
\]

$T[5] = 4$

**Goal:** For all $i$, output, $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

The Hamming distance is the number of mismatches...

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]
Pattern matching with mismatches

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$P$</td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goal:** For all $i$, output, $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches...*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input**: A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

- $T$: $a b c d a b a a d a c a a a$
- $P$: $a b d a$

**Goal**: For all $i$, output $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

The Hamming distance is the number of mismatches...

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$
Pattern matching with mismatches

**Input** A text string \( T \) (length \( n \)) and a pattern string \( P \) (length \( m \))

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
T & a & b & c & d & a & b & a & a & d & a & c & a & a \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
P & a & b & d & a \\
\end{array}
\]

\[\text{Ham}(8) = 3\]

**Goal:** For all \( i \), output, \( \text{Ham}(i) \), the Hamming distance between \( P \) and \( T[i \ldots i + m - 1] \)

*The Hamming distance is the number of mismatches...*

i.e. the number of distinct \( j \) such that \( P[j] \neq T[i + j] \)
**Pattern matching with mismatches**

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>$P$</td>
<td>a</td>
<td>b</td>
<td>d</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goal:** For all $i$, output, $\text{Ham}(i)$, the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches...*

i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$

- A naive algorithm takes $O(nm)$ time
Pattern matching with mismatches

**Input** A text string $T$ (length $n$) and a pattern string $P$ (length $m$)

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|
| $T$ | a | b | c | d | a | b | a | a | d | a | c | a | a |
| $P$ | a | b | d | a |

Hamming distance: $\text{Ham}(8) = 3$

**Goal:** For all $i$, output, Ham($i$), the Hamming distance between $P$ and $T[i \ldots i + m - 1]$

*The Hamming distance is the number of mismatches...*  
i.e. the number of distinct $j$ such that $P[j] \neq T[i + j]$

- A naive algorithm takes $O(nm)$ time

*Can we do better using cross-correlations...*
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

\[
\begin{array}{cccccccccc}
\text{0} & \text{1} & \text{2} & \text{3} & \text{4} & \text{5} & \text{6} & \text{7} & \text{8} & \text{9} & \text{10} & \text{11} & \text{12} \\
T & \begin{array}{cccccc}
a & b & c & d & a & b & a & a & d & a & c & a & a \\
\end{array} \\
P & \begin{array}{ccc}
a & b & d & a \\
\end{array} \\
\hline
m
\end{array}
\]
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

\[ T \]
\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
a & b & c & d & a & b & a & a & d & a & c & a & a \\
\end{array}
\]

\[ P \]
\[
\begin{array}{ccc}
a & b & d & a \\
\hline
m
\end{array}
\]

Replace all \( a \) symbols with 1 and everything else with 0.
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
T & a & b & c & d & a & b & a & a & d & a & c & a & a \\
\hline
P & a & b & d & a \\
\hline
m
\end{array}
\]

Replace all \(a\) symbols with 1 and everything else with 0.
Imagine that the alphabet contains only a small number of different symbols, which we consider individually…

Replace all $a$ symbols with 1 and everything else with 0
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually . . .

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
T & 1 & b & c & d & 1 & b & 1 & 1 & d & 1 & c & 1 & 1 \\
P & 1 & b & d & 1 \\
\hline
m
\end{array}
\]

Replace all a symbols with 1 and everything else with 0
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

Replace all $\alpha$ symbols with 1 and everything else with 0
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

Replace all α symbols with 1 and everything else with 0.
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

\[
\begin{array}{cccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
T & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
P & 1 & 0 & 0 & 1 \\
\end{array}
\]

*Replace all α symbols with 1 and everything else with 0*
Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

![Diagram of strings Ta and Pa]

Replace all \( a \) symbols with 1 and everything else with 0

We denote these new strings \( T_a \) and \( P_a \) and consider...
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually…

\[
\begin{array}{ccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
T_{a} & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
\end{array}
\]

\[
P_{a} & 1 & 0 & 0 & 1 \\
\hline
\end{array}
\]

\[
\text{Replace all } a \text{ symbols with } 1 \text{ and everything else with } 0
\]

We denote these new strings \( T_{a} \) and \( P_{a} \) and consider…

\[
(T_{a} \otimes P_{a})[i] = \sum_{j=0}^{m-1} P_{a}[j] T_{a}[i + j]
\]
It’s a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

\[
\begin{align*}
T_a &= \begin{array}{cccccccccccc}
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
\end{array} \\
P_a &= \begin{array}{cccc}
1 & 0 & 0 & 1 \\
\end{array}
\end{align*}
\]

Replace all \(a\) symbols with 1 and everything else with 0

We denote these new strings \(T_a\) and \(P_a\) and consider...

\[
(T_a \otimes P_a)[i] = \sum_{j=0}^{m-1} P_a[j]T_a[i + j]
\]

1 iff \(P[j] = T[i+j] = a\)
Imagine that the alphabet contains only a small number of different symbols, which we consider individually.

Replace all \( a \) symbols with \( 1 \) and everything else with \( 0 \).

We denote these new strings \( T_a \) and \( P_a \) and consider...

\[
(T_a \otimes P_a)[i] = \sum_{j=0}^{m-1} P_a[j] T_a[i+j]
\]

This is the number of matching \( a \)s at the i-th alignment.
It's a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

Replace all $a$ symbols with 1 and everything else with 0

We denote these new strings $T_a$ and $P_a$ and consider...

$$(T_a \otimes P_a)[i] = \sum_{j=0}^{m-1} P_a[j] T_a[i+j]$$

This is the number of matching $a$'s at the $i$-th alignment.
It's a small alphabet after all

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

Replace all \( a \) symbols with 1 and everything else with 0

We denote these new strings \( T_a \) and \( P_a \) and consider...

\[
(T_a \otimes P_a)[i] = \sum_{j=0}^{m-1} P_a[j] T_a[i+j]
\]

This is the number of matching \( a \)s at the i-th alignment.

which we can compute (for all \( i \)) in \( O(n \log m) \) time
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size

**Algorithm Summary**

Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$
Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$)
For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i].$$
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size

**Algorithm Summary**

Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$

Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$)

For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i].$$
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size

**Algorithm Summary**

Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$
Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$)
For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i].$$
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time.

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size.

**Algorithm Summary**

1. Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$.
2. Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$).
3. For every $i$, compute,

   $$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i].$$

   mismatches = $m$ - matches
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size

**Algorithm Summary**

- Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$
- Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$)
- For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i].$$
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size

**Algorithm Summary**

- Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$ ($O(n|\Sigma| \log m)$ time)
- Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$)
- For every $i$, compute,

$$
\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i] .
$$
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time.

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size.

**Algorithm Summary**

1. Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$ (O($n|\Sigma| \log m$) time).
2. Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$) (O($n|\Sigma| \log m$) time).
3. For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i].$$
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size

**Algorithm Summary**

Construct $T_{\sigma}$ and $P_{\sigma}$ for every symbol $\sigma$ in $\Sigma$ \hspace{1cm} \((O(n|\Sigma| \log m) \text{ time})\)

Compute $T_{\sigma} \otimes P_{\sigma}$ (for every symbol $\sigma$ in $\Sigma$) \hspace{1cm} \((O(n|\Sigma| \log m) \text{ time})\)

For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_{\sigma} \otimes P_{\sigma})[i]. \hspace{1cm} (O(n|\Sigma|) \text{ time})$$
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size

Algorithm Summary

Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$  
($O(n|\Sigma| \log m)$ time)

Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$)  
($O(n|\Sigma| \log m)$ time)

For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i].$$

($O(n|\Sigma|)$ time)

This takes $O(n|\Sigma| \log m)$ total time (and $O(n)$ space)
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size

Algorithm Summary

Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$ \hspace{1cm} (O(n|\Sigma| \log m) time)
Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$) \hspace{1cm} (O(n|\Sigma| \log m) time)
For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i].$$ \hspace{1cm} (O(n|\Sigma|) time)

This takes $O(n|\Sigma| \log m)$ total time (and $O(n)$ space)

However, $|\Sigma|$ could be as big as $m$...
It’s a small alphabet after all

We saw how to find all matches with a single symbol in $O(n \log m)$ time.

Let $\Sigma$ denote the set of alphabet symbols and $|\Sigma|$ be its size.

**Algorithm Summary**

Construct $T_\sigma$ and $P_\sigma$ for every symbol $\sigma$ in $\Sigma$ \( (O(n|\Sigma| \log m) \text{ time}) \)

Compute $T_\sigma \otimes P_\sigma$ (for every symbol $\sigma$ in $\Sigma$) \( (O(n|\Sigma| \log m) \text{ time}) \)

For every $i$, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_\sigma \otimes P_\sigma)[i]. \quad (O(n|\Sigma|) \text{ time})$$

This takes $O(n|\Sigma| \log m)$ total time (and $O(n)$ space).

However, $|\Sigma|$ could be as big as $m$...

in which case, it is worse than the naive method!
Conclusions

- Pattern matching with wildcards can be computed in $O(n \log m)$ time.
- Pattern matching with mismatches can be computed in $O(n|\Sigma| \log m)$ time.
Conclusions

- Pattern matching with wildcards can be computed in $O(n \log m)$ time.
- Pattern matching with mismatches can be computed in $O(n|\Sigma| \log m)$ time.

“Maybe if I had paid more attention in Data Structures and Algorithms,
I would find this comic funny”

- Anonymous Bristol CS student (posted on Facebook)